

# Artificial Neural Networks on Reconfigurable Meshes

Jing-Fu Jenq CS Department, Tennessee State University, jenq01@harpo.tnstate.edu  
Wing Ning Li CS Department, University of Arkansas, wingning@uafsysb.uark.edu

**Abstract** :Artificial neural networks(ANN) have been used successfully in applications such as pattern recognition, image processing, automation and control. Majority of today's applications use backpropagate feedforward ANN. In this paper, two methods of  $P$  pattern  $L$  layer ANN learning on  $n \times n$  RMESH have been presented. One required memory space of  $O(nL)$  but conceptually is simpler to develop and the other uses pipelined approach which reduces the memory requirement to  $O(L)$ . Both of these algorithms take  $O(PL)$  time and are optimal for RMESH architecture.

**Keywords** :Artificial Neural Networks, Reconfigurable Mesh Algorithms, Parallel Algorithms.

## 1. Introduction

The artificial neural networks(ANN) are very useful in applications such as pattern recognition, image processing, automation and control. Majority of today's applications use backpropagate feedforward ANN. In this paper, we will focus on the algorithm development of backpropagate feedforward ANN on reconfigurable mesh with buses[9]. An ANN consists of three types of layers in general, i.e., input layer, hidden layers and output layer. The total number of layers can be different for different applications. There are nodes(neurons) on each layer which connect to the input from the input interface or from the nodes of previous layer, and to output layer or the next layer. Associated with the arcs which connect two nodes are weights. The weights of the ANN can be changed during a learning process. For instance, during a supervised learning, input patterns are presented to the ANN and the actual output is compared with an expected output. The error, which is the discrepancy between the actual output and expected output can then be used as a guide to modify the weights. The weight modifications can be done through backward propagation of errors from the last layer back to the input layer. Let  $W_{ij}$  be the weight connect node  $i$  from layer  $l$  to node  $j$  of layer  $l+1$ . Assuming that the output of node  $i$  in the forward phase is  $o_i$  and output from node  $j$  is  $o_j$ . If the learning rate is  $\gamma$ . The backpropagation formula to modify the weight  $W_{ij}$  would be as the following

$$\Delta W_{ij} = \gamma o_i o_j (1 - o_j) \beta_j,$$

$$\beta_j = \sum_k W_{jk} o_k (1 - o_k) \beta_k, \text{ for nodes in hidden layers,}$$

$$\beta_z = d_z - o_z \text{ for nodes in the output layer, where } d_z \text{ is the desired output and } o_z \text{ is the observed output. Let us call } \beta_z \text{ the error.}$$

In the forward phase, the output of a node will be multiplied with the weights of its outgoing arcs. Each one of these multiplication results contributes to the input of the nodes in the next layer. To determine the output of a node, an

activation function can be used which will generate an output by taking into account of all the contributions from its input links. One simple method is to add together all its input and then apply a threshold function to generate the output for that node. The collection of these outputs on each layer are called the Activation values of that layer. In the backward phase, the errors and all the Activation values for a particular pattern is used to determine how much the weights shall be modified.

The learning process is an iterative operation. In the forward phase, the input patterns are fed into the input layer. Each hidden layer does the computation and forwards the Activation values to the next layer. The output layer computes the errors. These errors will then be backpropagated, by using the backpropagation formula mentioned earlier back to the input layer. The modification of the weights can be done, during backward phase, on the arcs connect the nodes of the layers. The weights can be modified each time a pattern has been processed, or when a group of patterns has been processed, or when all the patterns have been processed. For the last two approaches, the weight changes can be added together and weights are updated after all patterns has been presented to the ANN. For parallel implementation concerns, the modification of weights in general will be done after one iteration of feeding all patterns. The learning process will be terminated when the total error is within an acceptable level or when the limit of the number of iterations has been reached. The learning process of an ANN is very time consuming. For an ANN with  $L$  layers and  $n$  nodes each, assuming complete connection among the nodes between two adjacent layers, and  $P$  patterns, the run time for one iteration would be  $O(PLn^2)$ . In this paper, we assume  $P \gg n$ .

El-Amawy and Kulasinghe mapped feedforward ANNs onto multiple bus systems[5]. Kumar, et. al. used well known checkboarding method to map the backpropagation algorithms on hypercube and related architectures[11]. Malluhi et al. presented algorithm mapping of ANN on hypercube machine by using pipelined approach[16]. Lin, Prasanna and Przytula described parallel implementation of neural networks on fine-grain SIMD MESH[15]. Chu and Wah mapped feedforward ANN on message passing multicomputers[2]. Park and Thornbrugh trained the ANN by partitioned the patterns among various processors, where coarse grain parallelism is used[18]. Chin et al. showed how the systolic array implementation of feedforward neural nets can run on MasPar MP-1SIMD machine[1]. Hiraiwa et al., mapped the backpropagation neural net on to a two level pipeline RISC processor array[8]. Ramacher presented a neurocomputer architecture concept which can be easily adapted to the application under consideration[19]. For systolic array implementation, Kung presented the ANN implementation by using programmable systolic ring arrays[12]. Chung et al. presented 2-D systolic array[3]. Lehmann presented a generic systolic array building block for ANNs[14]. Lanser and Lehmann developed an analog CMOS chip for neural networks with arbitrary topologies[13]. Hammerstrom discussed a VLSI architecture called X1[6]. Clarkson et al., presented VLSI pRAM chip to approximate biological

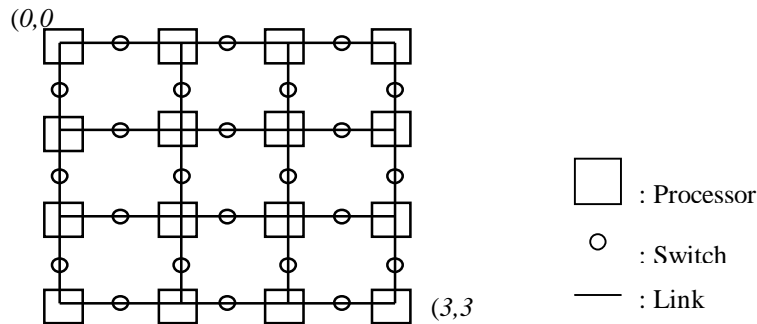
neuron[4]. A survey of using and designing massively parallel computers for artificial networks can be found in [17].

In section 2, the architecture of reconfigurable mesh with buses(RMESH) is described. Section 3 collects the basic operations that are used in the development of the algorithms. Two cases are discussed in section 4, one with memory space of  $O(Ln)$  and the other  $O(L)$ . Section 5 consists of the conclusions of the report.

## 2. Reconfigurable Mesh with Buses (RMESH)

RMESH employs a reconfigurable bus to connect all processors. Figure 1 shows a  $4 \times 4$  RMESH. The important features of RMESH are:

1. An  $N \times M$  RMESH is a 2-dimensional mesh connected array of processing elements (PEs). Each PE in the RMESH is connected to a broadcast bus which is itself constructed as an  $N \times M$  grid. The PEs are connected to the bus at the intersections of the grid. Each processor has up to four bus switches (Figure 1) that can be used to reconfigure the bus into subbuses. The ID of each PE is a pair  $(i, j)$  where  $i$  is the row index and  $j$  is the column index. The ID of the upper left corner PE is  $(0,0)$  and that of the lower right one is  $(N-1, M-1)$ .



**Figure 1** A  $4 \times 4$  RMESH

2. There are up to four switches associated with a PE are labeled E (east), W (west), S (south) and N (north). Two PEs can simultaneously set or unset a particular switch as long as the settings do not conflict. The broadcast bus can be subdivided into subbuses by opening some of the switches.
3. Only one processor can put data onto a given sub bus at any time.
4. In unit time, data put on a subbus can be read by every PE connected to it. If a PE is to broadcast a value in register I to all of the PEs on its subbus, then it uses the command broadcast(I).
5. To read the content of the broadcast bus into a register R, the statement  $R = \text{content}(\text{bus})$  is used.
6. Row buses are formed if each processor disconnects (opens) its S switch and connects (closes) its E switch. Column buses are formed by disconnecting the E switches and connecting the S switches.

### 3. Basic Data Manipulation Operations

In this section we define several basic data manipulation algorithms[9] for RMESH multicomputers. These are used in the next section to develop algorithms for backpropagation feedforward ANN.

#### 3.1 Diagonalization

This operation will move the specific row (column) elements to the diagonal position of a specified window which contains that row (column). It takes  $O(I)$  time.

#### 3.2 Data Sum

Initially, each PE of the  $N \times N$  RMESH has an  $A$  value. Each PE is to sum up the  $A$  values of all the  $N^2$  PEs and put the result in its  $B$  variable. I.e., following the data sum operation we have :  $B(i, j) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} A(i, j), 0 \leq i, j < N$ . This can be done in  $O(\log N)$  time

#### 3.3. Shift

Each PE has data in its  $A$  variable that is to be shifted to the  $B$  variable of a processor that is  $s$ ,  $s > 0$ , units to the right but on the same row. One of the variant of the shift is the operation of circular shift which performs shift with wrap around feature. These operations can be done in  $O(s)$  time. If  $s=1$  then the time becomes  $O(1)$ .

#### 3.4. Consecutive Sum

Assume that an  $N \times N$  RMESH is tiled by  $I \times M$  blocks ( $M$  divides  $N$ ) in a natural manner with no blocks overlapping. So, processor  $(i, j)$  is the  $j \bmod M$ 'th processor in its block. Each processor  $(i, j)$  of the RMESH has an array  $X[0..M-1](i, j)$  of values. If  $j \bmod M = q$ , then PE  $(i, j)$  is to compute  $S(i, j)$  such that  $S(i, j) = \sum_{r=0}^{M-1} X[q](i, (j/M) * M + r)$ . That is, the  $q$ 'th processor in each block sums the  $q$ 'th  $X$  value of the processors in its block. The consecutive sum operation can be done in a similar way for an  $M \times I$  block and its time complexity is  $O(M)$ .

### 4. Backpropagation on RMESH

Assuming the ANN considered has  $L$  layers and each layer consists of  $n$  nodes, and the dimension of RMESH is  $N \times N$ . Because the computation of backward phase is similar to forward phase. Let us consider basic procedures for forward phase.

First of all, let us consider an ANN with the number of nodes on each layer is  $n$  and  $n=N$ , Note that the dimension of RMEAH is  $N \times N$ . Further assume  $P \gg Ln$ ,  $Ln$  is the total number of nodes of the Neural network, and  $P$  is the total number of patterns involved in the training. Assuming that the PEs with indices  $(0,0)$

to  $(N-1, 0)$  are connected to the input interface of the system and can retrieve the pattern input and expected(desired) output from this interface. Let  $input(p,i)$  be the  $i$ th input value of pattern  $p$ . Further assume that each PE has  $L$  weights associated with it. The PE( $i,j$ ) holds ANN weights of  $W_j[k], 0 \leq i, j < N, 0 \leq k < L-1$ . The checkboarding method used by Kumar et. al. [11] can be implemented on RMESH and the complexity is  $O(PL \log N)$  for one iteration of learning process[10]. Two new methods are developed which improve the time complexity to  $O(PL)$ . We will first present procedure which requires  $O(nL)$  space and later present another procedure which reduces the space to  $O(L)$ .

With  $O(nL)$  space being available, we can partition the  $P$  patterns into  $\lceil P/n \rceil$  groups with each group holds a maximum of  $n$  patterns. Instead of processing one pattern at a time, we process a group of patterns at a time. The procedure is shown in Figure 2. Since the backward phase computation is similar to the forward phase, we omit the detail here. Without loss of generality, let us assume  $n$  divides  $P$ .

Step S1.1.0 reads in the input data and the desired output data of the patterns. The values will then be distributed to the processors that need them through row buses and the broadcast operations(Step S1.1.1 to S1.1.4). The operations will be done  $n$  times for all the  $n$  patterns in this group. Since all these operations take  $O(1)$  time, therefore the total time complexity for  $n$  patterns is  $O(n)$  and for all the  $P$  patterns would be  $O(P)$ . Step 1.2 does the forward phase computations. Step S1.2.2 sums up the results of multiplying the activation values by the weights toward the nodes of next layer in the chain of the network. It takes  $O(n)$  time for  $n$  processors in the row. Step 1.2.3 applies threshold function  $F$  to the total input to generate and prepare the activation values for the next layer computation. Steps 1.2.5.1 and S1.2.5.2 broadcast the activation values to the processors which must have them. They take once again another  $O(n)$  time. Since there are  $O(L)$  layers, the total complexity would be  $O(nL)$  for step S1.2. Because step S1.2 will be done  $O(P/n)$  time, so the total time complexity would be  $O((P/n) \cdot n \cdot L) = O(PL)$  which is optimal for RMESH architecture. Yet the overall space complexity is  $O(nL)$ .

Now, let's consider the case when the available memory space is limited to  $O(L)$ . The procedure in Figure 3 presents the computation of activation outputs for a particular layer  $k$ . This is done by using a pipelined approach. Let us assume the activation values of  $n$  patterns are distributed among the processors as shown in Figure 4. Here four patterns are considered. So, a total of sixteen activation values are presented at the time of forward computation. This procedure takes  $O(n)$  time to compute the result. As discussed earlier the input activation values have to broadcast to all the processors which must have them for performing the multiplication with the weights which are distributed to each processor initially. The weights associated with the PEs will be broadcasted by using the pipelined method as shown in Figure

5. Here seven broadcasts are needed which are numbered as 0 to 6. To broadcast these sixteen weights. At iteration zero  $W_{00}$  will be broadcast through row bus to processors which need them to perform the multiplication with the Activation values. More specifically,  $A_{00} \times W_{00}$ ,  $A_{10} \times W_{00}$ ,  $A_{20} \times W_{00}$ ,  $A_{30} \times W_{00}$  will be computed at this iteration. At iteration one,  $W_{01}$  of row 1 and  $W_{10}$  of row 2 will be broadcasted by using row buses. At iteration two, the broadcasting of  $W_{02}$ ,  $W_{11}$ ,  $W_{20}$ , ... Finally at iteration six, the broadcasting of  $W_{33}$ .

---

```

S0   for  $k = 1$  to  $L-1$  do  $\Delta W[k] = 0$ ;
S1   for  $m = 0$  to  $P/n - 1$  do
      begin
S1.1 for  $p = 0$  to  $n-1$  in  $m$ th group do
      begin
S1.1.0 PE( $i,0$ ) read  $input[p,i]$  and  $output[p,i]$ , from  $(mn+p)$ th
      pattern, for  $0 \leq i < N$ .
S1.1.1 Setup row bus, PE( $i,0$ ) broadcast  $(input[p,i])$ ,  $0 \leq i < N$ 
S1.1.2  $A[p,0](i, j) = content(bus)$ ,  $0 \leq i, j < N$ 
S1.1.3 PE( $i,0$ ) broadcast  $(output[p,i])$ ,  $0 \leq i < N$ 
S1.1.4  $desired\_output[p](i,j) = content(bus)$ , for  $i = j$ 
      end
S1.2 for  $k = 0$  to  $L-2$  do
      begin
S1.2.1 for  $p = 0$  to  $n-1$  do
           $T[p](i, j) = A[k](i, j) \times W[k](i, j)$ , for  $0 \leq i, j < N$ 
S1.2.2 Consecutive_sum on  $T[p](i, j)$ 
S1.2.3 for  $p = 0$  to  $n-1$  do
           $A[p, k+1](i,j) = F(T[p](i,j))$ ,  $0 \leq i, j < N$ 
S1.2.4 Setup row bus
S1.2.5 for  $p = 0$  to  $n-1$  do
          begin
S1.2.5.1 Broadcast  $(A[p, k+1](i,j))$ , for PEs with  $i = j$ 
S1.2.5.2  $A[p, k+1](i, j) = content(bus)$ ,  $0 \leq i, j < N$ 
          end (* of compute activation values for n patterns *)
      end (* of forward computation *)
S1.3 for  $p = 0$  to  $n-1$  do
       $\beta[p, L-1](i, j) = desired\_output[p](i, j) - A[p, L-1](i, j)$ , for  $i = j$ 

```

---

**Figure 2** ANN Forward phase computation when  $O(nL)$  space is available  
 In Figure 3, Steps 1.1 to 1.3 broadcast weights through row buses. Step 1.4 computes  $W \times A$  on each node. Steps 1.5 and 1.6 determine which nodes are supposed to send and/or receive tokens. If a PE needs to send a token then at Step 1.7 a shift operation will be performed to send the accumulated  $W \times A$  to the PE immediately below it. Step S2 is slightly different from S1. At this stage the new activation values for the

next layer will be generated. These values will be stored to the appropriate PEs through column buses and broadcasting operations. Steps 2.4 to Step 2.11 are similar to Step1.1 to Step1.8. Finally at Step 3, the last activation values are stored to the PEs of the bottom row which do not involve any broadcast operation.

---

Procedure Forward\_Computation(layer)

**begin**

$D(i,j) = 0;$

S1 **for**  $k=0$  **to**  $n-2$  **do**

**begin**

S1.1 Setup row bus

S1.2 Broadcast  $W(i,j)$ , for  $i + j = k$

S1.3  $C(i, j) = \text{content}(\text{bus})$ , for  $i \leq k$

S1.4  $T(i, j) = C(i, j) \times A[\text{layer}](i, j) + D(i, j)$ , for  $i \leq k$

S1.5 if  $i \leq k$  then  $\text{send\_token}(i,j) = \text{true}$

S1.6 if  $(i > 0)$  and  $(i \leq k + 1)$  then  $\text{receive\_token}(i,j) = \text{true}$

S1.7 if  $\text{send\_token}(i,j)$  then  $\text{shift}(\text{DOWN}, T, 1)$

S1.8 if  $\text{receive\_token}(i,j)$  then  $D(i, j) = \text{content}(\text{bus})$

**end**

/\* at this point, row  $n-1$  has first completed activation value of pattern ) for next layer \*/

S2 **for**  $k = n-1$  **to**  $2(n-1)$  **do**

**begin**

S2.1 Setup column bus

S2.2 PE( $n-1,j$ ) broadcast  $(T(i,j))$

S2.3  $A[\text{layer}](k - n - 1) = \text{content}(\text{bus})$

S2.4 Setup row bus

S2.5 broadcast  $W(i, j)$  for  $i + j = k$

S2.6  $C(i, j) = \text{content}(\text{bus})$ , for  $i > (k + 1)\%n$

S2.7  $T(i, j) = C(i, j) \times A[\text{layer}](i, j) + D(i, j)$ , for  $i > (k + 1)\%n$

S2.8 if  $(i > (k + 1)\%n)$  and  $(i \neq n - 1)$  then  $\text{send\_token}(i,j) = \text{true}$

S2.9 if  $(i > (k + 2)\%n)$  then  $\text{receive\_token}(i,j) = \text{true}$

S2.10 if  $\text{send\_token}(i,j)$  then  $\text{Shift}(\text{DOWN}, T, 1)$

S2.11 if  $\text{receive\_token}(i,j)$  then  $D(i, j) = \text{content}(\text{bus})$

**end**

S3  $A[\text{layer} + 1](n - 1, j) = D(i, j)$  /\* last row \*/

---

**end;** (\* of procedure Forward\_Computation \*)

**Figure 3** Computation of Activation values for layer  $k+1$  give the activation values at layer  $k$

Since the shift operation of Step 1.7 and Step 2.10 take  $O(1)$  time, the total time complexity is  $O(n)$  which handles the computation of one layer. Therefore for a  $L$  layer ANN, the time complexity would be  $O(nL)$ . Because the  $P$  training patterns can

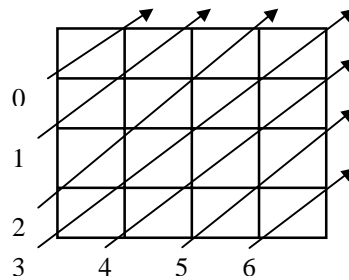
be partitioned into  $P/n$  groups and processed separately, the total time complexity is therefore  $O(PL)$ . The space requirement is easily seen to be  $O(L)$ .

## 5. Conclusions

Two methods of  $P$  pattern  $L$  layer ANN learning on  $n \times n$  RMESH have been presented. These two RMESH algorithms have the same time complexity of  $O(PL)$  which is optimal for RMESH architecture.. One with space complexity of  $O(nL)$  but conceptually is simpler to develop and the other uses the pipelined approach and reduces the space complexity to  $O(L)$ .

A00	A10	A20	A30
A01	A11	A21	A31
A02	A12	A22	A32
A03	A13	A23	A33

**Figure 4** The distribution of Activation values of four patterns on RMESH for a particular layer  $l$



**Figure 5** The broadcasting of sixteen weight values through row bus by using pipelined approach

## 6. References

- [1] G. Chinn, K. A. Grajski, C. Chen, C. Kuszmaul, and S. Tomboulian, "Systolic Array implementations of Neural Nets on the MasPar MP-1 Massively Parallel Processor", International Conference Neural Networks, vol. 2, pp 169-173, San Diego, 1990
- [2] L. Chu and W. Wah, "Optimal mapping of Neural-Network Learning on Message-Passing Multicomputers", Journal of Parallel and Distributed Computing, vol. 14, pp-319-339, 1992
- [3] J. Chung, H. Yoon, and S. R. Maeng, "A systolic Array Exploiting the Inherent Parallelisms of Artificial Neural Networks", International Conference on Parallel Processing, vol. 1, pp 652-653, 1991

- [4] T. G. Clarkson, C. K. Ng, and Y. Guan, "The pRAM: An Adaptive VLSI Chip", IEEE Transactions on Neural Networks, vol. 4, no. 3, pp 408-411, May 1993
- [5] A. El-Amawy, and P. Kulasinghe, "Algorithmic Mapping of Feedforward Neural Networks onto Multiple Bus Systems", IEEE transactions on Parallel and Distributed Systems, vol. 8, pp130-136, Feb. 1997
- [6] D. Hammerstrom, "A VLSI Architecture for High-Performance, Low-Cost, On-chip Learning", International Joint Conference on Neural Networks, vol. 2, pp 537-543, 1990
- [7] S. Haykin, *Neural Networks, A Comprehensive Foundation*, IEEE Press, 1994
- [8] A. Hiraiwa, S. Kurosu, S. Arisawa and M. Ionue, "A Two Level pipeline RISC Processor Array for ANN", International Joint Conference on Neural Networks, Washington DC, vol. 2, pp137-140, 1990
- [9] J. Jenq and S. Sahni "Reconfigurable Mesh Algorithms for Fundamental Data Manipulation Operations", *Computing on Distributed Memory Multiprocessors*, NATO Series F, ed. F. Ozguner, Springer Verlag, 1993
- [10] J. Jenq and W. Li "Artificial Neural Networks on Reconfigurable Meshes", CSCI-TR-98-01 Department of Computer Science, University of Arkansas
- [11] V. Kumar, S. Shekhar, and M. Amin, "A Scaleable Parallel Formulation of the Backpropagation Algorithm for Hypercubes and Related Architectures", IEEE Transactions on Parallel and Distributed Systems, vol. 5, no. 10, pp 1073-1090, Oct. 1994
- [12] S. Y. Kung, "Parallel Architectures for Artificial Neural Nets", International Conference on Systolic Arrays, pp 163-174, 1988
- [13] J. Lansner and T. Lehmann, "An Analog CMOS Chip Set for Neural Networks with Arbitrary Topologies", IEEE Transactions on Neural Networks, vol. 4, no. 3, pp 441-444, May 1993
- [14] C. Lehmann, M. Viredaz, and F. Blayo, "A Generic Systolic Array Building Block for Neural Networks with on-Chip Learning", IEEE Transactions on Neural Networks, vol. 4, no. 3, pp 400-407, May 1993
- [15] W. Lin, V. K. Prasanna, and K. W. Przytula, "Algorithmic Mapping of Neural Network Models onto Parallel SIMD Machines", IEEE Transactions on Computers, vol. 40, no. 12, pp 1390-1401, Dec. 1991
- [16] Q. M. Malluhi, M. Bayoumi, and T. R. N. Rao, "Efficient mapping of ANNs on Hypercube Massively Parallel Machines", IEEE Transactions on Computers, vol. 44, no. 6, pp769-779, June 1995
- [17] T. Nordstrom and B. Svensson, "Using and Designing Massively Parallel Computers for Artificial Neural Networks", Journal of Parallel and Distributed Computing, vol. 14, pp 260-285, 1992
- [18] K. Parker and A. Thornbrugh, "Parallelized Back-Propagation Training and Its Effectiveness", vol. 2, International Conference on Neural Networks, Washington DC, vol. 2, pp179-182, Jan. 1990
- [19] U. Ramacher, "SYNAPSE-A Neuralcomputer that Synthesizes Neural Algorithms on a Parallel Systolic Engine", Journal of Parallel and Distributed Computing, vol. 14, pp 306-318, 1992